# An example of Bayesian inference for Perceptron learning

Bert Kappen,
SNN, Radboud University Nijmegen, 2005.

March 10, 2014*

This is a computer demo that illustrates Bayesian inference for the Perceptron learning task, as explained in Mackays book Ch. 39 and 41, using MCMC sampling methods from Ch. 29 and 30. To start these computer excersize, download the `mcmc_mackay.tar.gz` file from the course website, and extraxt it, for example by executing the following commands:

```
gunzip mcmc_mackay.tar.gz
tar xvf mcmc_mackay.tar
```

This creates the directory `mcmc_mackay`. Go to this directory, start Matlab, and we are ready to go!

This directory contains a number of Matlab programs to illustrate Bayesian inference applied to the Perceptron, as discussed in Chapters 39 and 41, using some of the MCMC methods as discussed in Chapter 29. It illustrates the following issues as introduced in the Chapters of MacKay:

- the concept of weight space as shown in figure 39.3
- learning of a classification task by gradient descent as shown in figure 39.4
- learning with weight decay as shown in figure 39.6
- the difference between maximum likelihood and the bayesian posterior distribution as shown in figure 41.1
- the use of the Metropolis algorithm to sample the posterior distribution and the Bayesian solution to the classification problem as shown in figure 41.2

All illustrations use the same data set which is a classification problem in two dimensions. The data points are given in the file `x.ext` and their class labels are given in the file `t.ext`.

Startup Matlab by simply typing `matlab`. Programs within Matlab are started by simply typing the file name without `.m` extension. The programs can (and should!) be edited to see how different options affect the performance. Use your own favorite editor.

*updated by Sep Thijssen in 2014

Please write a short report in which you answer all the questions posed in the exercises below. Preferred format is Portable Document Format (.pdf) or plain text (.txt). Also please hand in source code files that you made or edited to perform the exercises, and make sure that these source files actually run without error and within a reasonable amount of time (in the order of 10 seconds or so).

**Exercise 1.** The program `WeightSpace.m` shows the data of the two classes as dots in the two-dimensional plane. Note, that $x_1 = 1$ for all data points to allow easy encoding of the threshold in the inner product $x \cdot w = x_1 w_1 + x_2 w_2 + x_3 w_3$. Thus the data shown are the $x_2$ and $x_3$ values. In addition it shows the perceptron classifier with weights $(w_1, w_2, w_3) = (10, -1, -1)$. By changing the weights (in the file `WeightSpace.m`), observe how the classifier is changed. What is the effect of changing $w \to -w$? What is the effect of scaling w by some positive number? Find the weights that best fit the classification problem.

**Exercise 2.** The program `GradDesc.m` is a very simple gradient descent learning algorithm that illustrates learning with and without weight decay. The program contains the following steps:

The program first loads the data. Then it sets the number of learn iterations $L$, the value of the weight decay parameter $\alpha$ (in the file `alpha`), and the learning step size $\eta$ (`eta`). The weights and threshold are initialized to zero, or some other value. The program executes $L$ learning steps. It implements the learning algorithm as described in Section 39.3. First, the outputs $y$ of the network with the current values of the weights $w$ are computed for all patterns $x$. Then, the errors $e$ are computed, which are the differences between the outputs $y$ and the desired outputs $t$. Since we use batch learning, the gradient $g$ is the sum of the contributions of all patterns. Finally, the weights are changed a small amount $\eta$ in the direction of the negative gradient ($\alpha = 0$). The learning algorithm will attempt to minimize the function $M(w)$ as given in Eq. 41.1. The values of $w$ and $M(w)$ at each iteration are stored in `w_all` and `M_all` and are plotted. Finally, the solution of the perceptron after learning is plotted.

Change the learning rate `eta`, the number of iterations `L` and the initial weights to see how it affect the learning process and the final solution.

Set `alpha=0.01` and `alpha=0.1` and see how it affects the final solution (hint: use `w=[-10;1;1]` as initial weight value).

**Exercise 3.** The program `FullBayes.m` illustrates the relative contribution of the likelihood and the prior on the posterior and their dependence on the amount of data. The value of the threshold is fixed to its (almost) optimal value -10. The program first loads the data and selects a subset of data points. Subsequently, the data points, the likelihood, and the posterior are plotted. Interpret the result when only one data point is selected. Observe how the results for various data sets depend on $\alpha$.

**Exercise 4.** The program `Metropolis.m` samples from the posterior. First the data is loaded. The variable `alpha` is the weight decay parameter and defines the prior distribution. The variables `sigma`, `L` and `Lburn` are parameters for the Metropolis algorithm. `L` sets the total number of iterations, `Lburn` defines the length of the burn-in period. Statistics is sampled after the burn-in period

only. The proposal distribution for the Metropolis Method is Normal with width `sigma`. The first plot shows the dynamics of the weights, including the burn-in period. The second plot shows the cost function $M(w)$ that is sampled. The third plot shows the sampling of $w_2$ and $w_3$ after burn in. Note, the similarity of this figure with the lower right plot in FullBayes. The fourth plot shows the Bayesian solution Eq. 41.18 approximated with Eq. 41.19. Try to find a good burn-in period for different initial weights, in particular it should burn-in well with the not-so-smart initial `w=[10,-1,-1]`. How does the effectiveness of the burn-in period depend on `sigma`? Try to get the best sampling with the minimum number of iterations, and find the optimal `sigma` to achieve this. Study the effect of `alpha` on the final solution, explain the result.

**Exercise 5.** The program `HMC.m` implements a Hamilton Monte Carlo method as described in Algorithm 30.1. In addition to the parameters `L` and `Lburn` the important parameters are: `Tau`, the number of leapfrog steps doing Hamilton dynamics, and `epsilon`, the Hamilton dynamics step size.

- Compare the Hamilton Monte Carlo method and standard Metropolis on the perceptron learning task.

As you may have concluded, for this problem, the difference in performance between Metropolis and HMC is not not spectacular. Instead, when the distribution is strongly elongated there is a difference. This is illustrated in the program `HMC1.m`. Consider the distribution $p(x_1, x_2) \propto \exp(-E(x_1, x_2))$, with $E = \frac{1}{2} x' A x$ and

$$A = \begin{pmatrix} 250.25 & -249.75 \\ -249.75 & 250.25 \end{pmatrix}$$

- Set `ALGORITHM_TYPE='metro'` to test the Metropolis algorithm. Running the algorithm will display the accepted sample points and print their total number. Adjust `sigma` to get the best performance. What happens when `sigma` is large or small?

- Set `ALGORITHM_TYPE='hmc'` to test the Hamilton Monte Carlo algorithm. Try different initializations for `x` and different combinations of `Tau`, `L` and `epsilon`.

- Compute the mean of this Gaussian distribution using both methods and compare the accuracy as a function of the computation time.